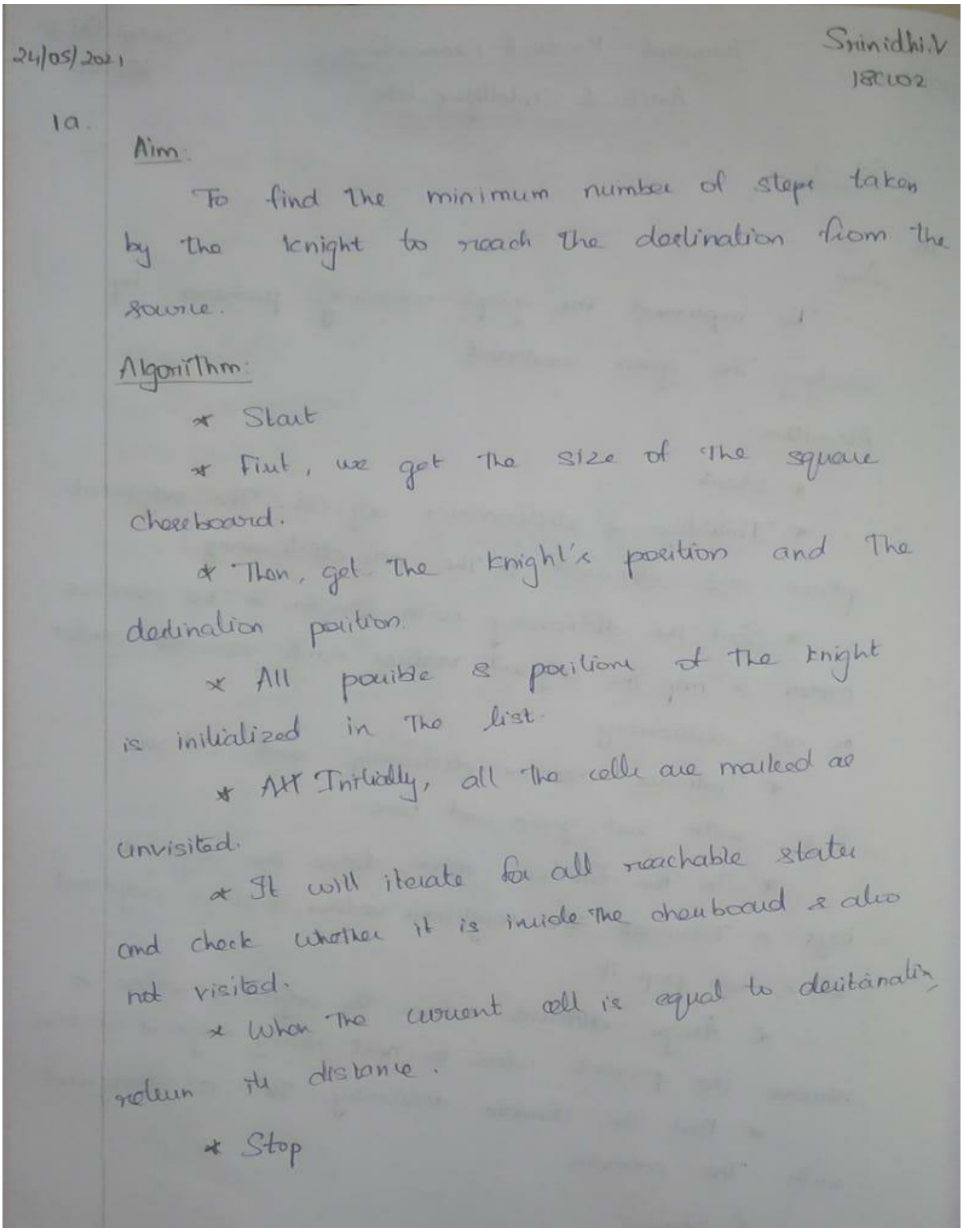
Terminal Practical Examination
Artificial Intelligence Lab

Name: Srinidhi V
Reg No. : 18C102
Slot : A
Set : 1



24/05/2021

Srinidhi.V
18C102

1a.

Aim:

To find the minimum number of steps taken by the knight to reach the destination from the source.

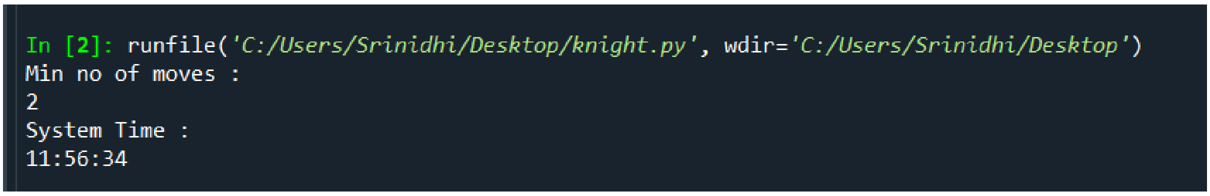
Algorithm:

- * Start
- * First, we get the size of the square chessboard.
- * Then, get the knight's position and the destination position.
- * All possible 8 positions of the knight is initialized in the list.
- * ~~All~~ Initially, all the cells are marked as unvisited.
- * It will iterate for all reachable states and check whether it is inside the chessboard & also not visited.
- * When the current cell is equal to desitanation, return its distance.
- * Stop

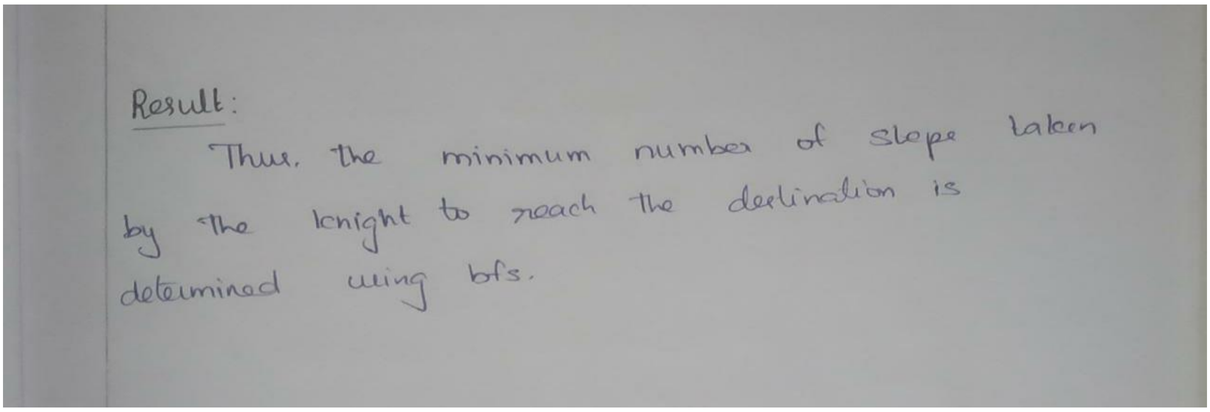
Program:

```
class cell:

    def __init__(self, x = 0, y = 0, dist = 0):
        self.x = x
        self.y = y
        self.dist = dist


def isInside(x, y, N):
    if (x >= 1 and x <= N and
        y >= 1 and y <= N):
        return True
    return False


def minStepToReachTarget(knightpos,
                         targetpos, N):


    dx = [2, 2, -2, -2, 1, 1, -1, -1]
    dy = [1, -1, 1, -1, 2, -2, 2, -2]

    queue = []



    queue.append(cell(knightpos[0], knightpos[1], 0))

    # make all cell unvisited
    visited = [[False for i in range(N + 1)]
                     for j in range(N + 1)]


    visited[knightpos[0]][knightpos[1]] = True


    while(len(queue) > 0):

        t = queue[0]
        queue.pop(0)
        if(t.x == targetpos[0] and
          t.y == targetpos[1]):
            return t.dist


        for i in range(8):

            x = t.x + dx[i]
            y = t.y + dy[i]

            if(isInside(x, y, N) and not visited[x][y]):
                visited[x][y] = True
                queue.append(cell(x, y, t.dist + 1))


if __name__=='__main__':
    N = 8
    knightpos = [3, 4]
    targetpos = [6, 3]
    print(minStepToReachTarget(knightpos,
                              targetpos, N))
```

Output:

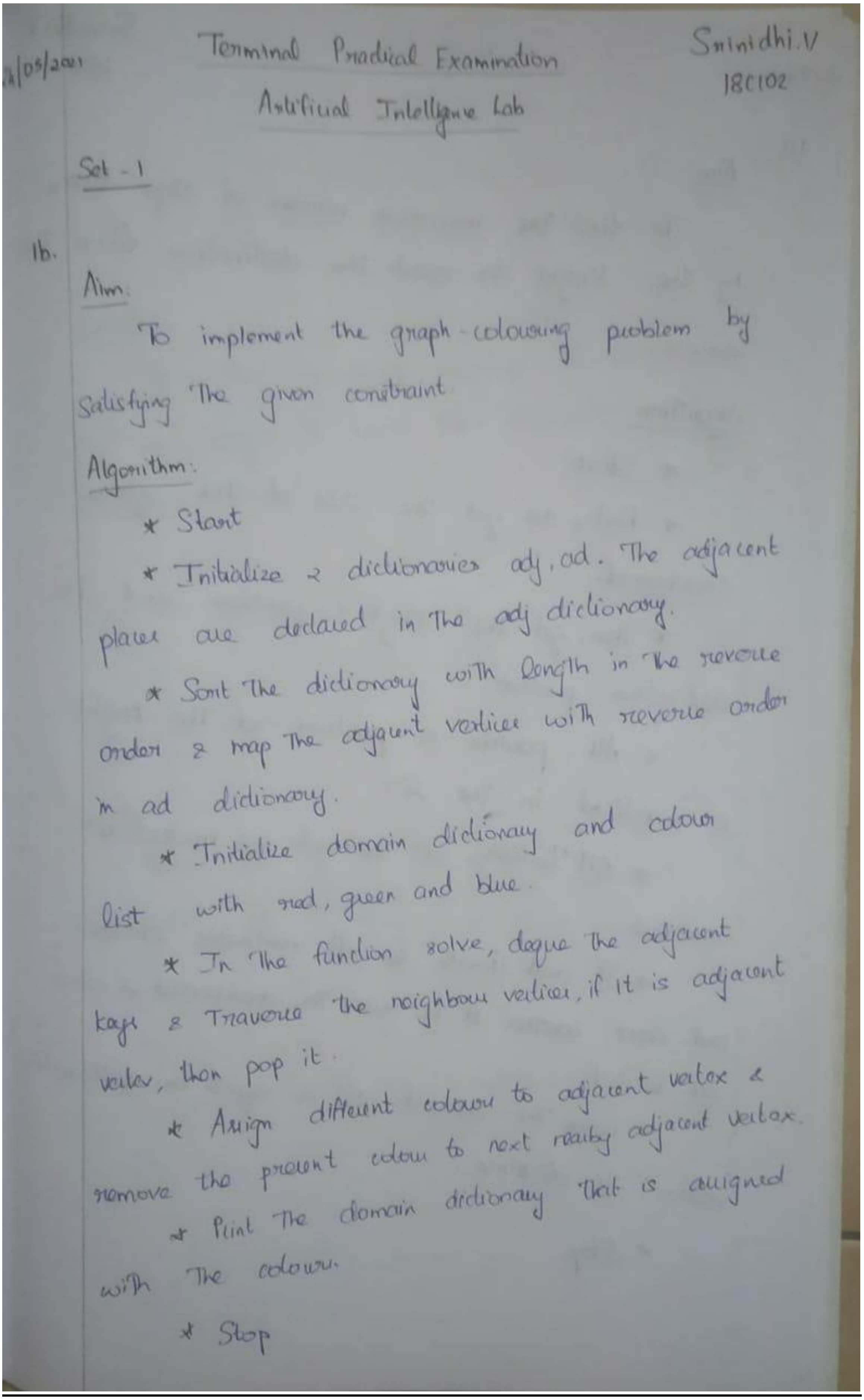


```
In [2]: runfile('C:/Users/Srinidhi/Desktop/knight.py', wdir='C:/Users/Srinidhi/Desktop')
Min no of moves :
2
System Time :
11:56:34
```



Result:

Thus, the minimum number of slope taken by the knight to reach the destination is determined using bfs.



2/05/2021

Terminal Practical Examination

Artificial Intelligence Lab

Srinidhi.V
18C102

Set - 1

1b.

Aim:

To implement the graph-colouring problem by satisfying the given constraint

Algorithm:

* Start

* Initialize 2 dictionaries adj, ad. The adjacent places are declared in the adj dictionary.

* Sort the dictionary with length in the reverse order & map the adjacent vertice with reverse order in ad dictionary.

* Initialize domain dictionary and colour list with red, green and blue.

* In the function solve, deque the adjacent keys & Traverse the neighbour vertices, if it is adjacent vertex, then pop it.

* Assign different colours to adjacent vertex & remove the present colour to next nearby adjacent vertex.

* Print the domain dictionary that is assigned with the colours.

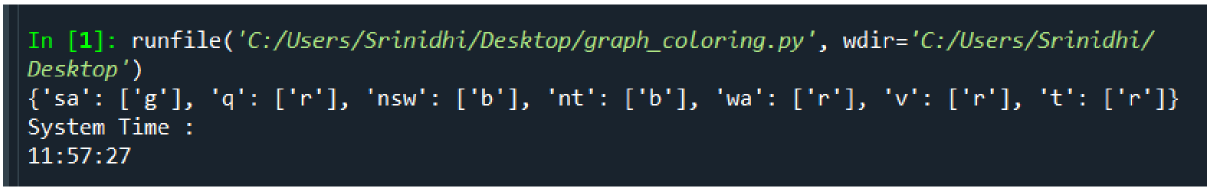
* Stop

Srinidhi V(18C102)

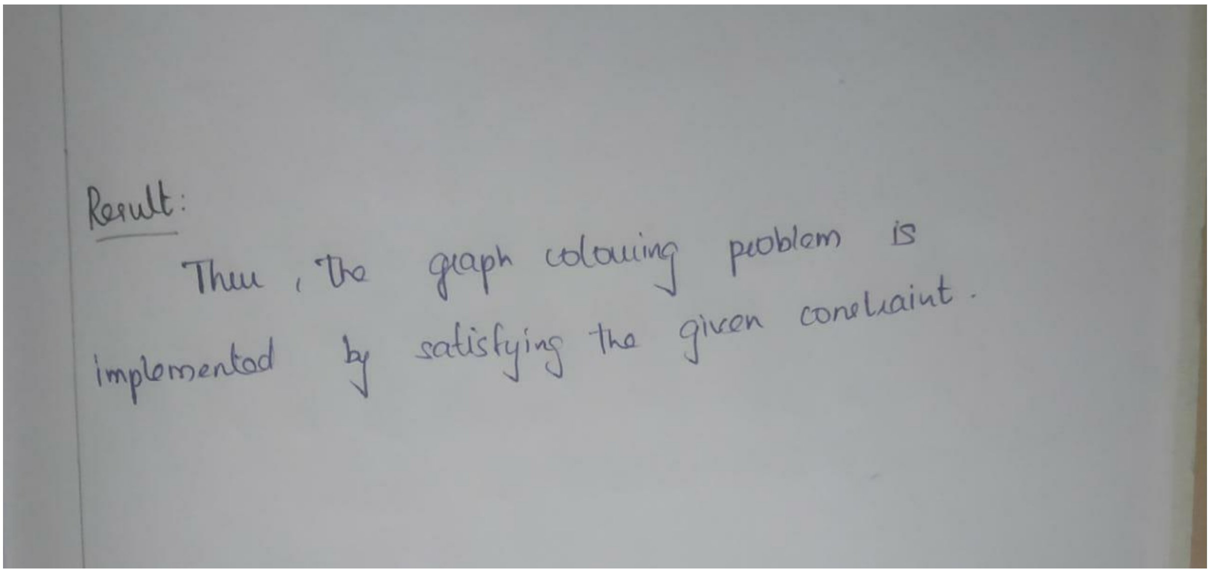
Program:

```
from collections import deque

def solve(domain, adj):
    q = deque(list(adj.keys()))

    while(q):
        curr_node = q.popleft()
        domain[curr_node] = list(domain[curr_node][0])

        for dest_node in adj[curr_node]:
            colors = domain[dest_node]
            if(domain[curr_node][0] in colors):
                colors.remove(domain[curr_node][0])


if __name__ == "__main__":
    adj, ad = dict(), dict()
    adj['q'] = ['nt', 'nsw', 'sa']
    adj['sa'] = ['wa', 'nt', 'q', 'nsw', 'v']
    adj['nsw'] = ['sa', 'q', 'v']
    adj['nt'] = ['wa', 'sa', 'q']
    adj['wa'] = ['nt', 'sa']
    adj['v'] = ['sa', 'nsw']
    adj['t'] = []
    temp = sorted(adj, key = lambda ele : len(adj[ele]), reverse = True)
    for ele in temp:
        ad[ele] = adj[ele][:]
    domain =  dict()
    colors = ['r', 'g', 'b']
    for key in ad.keys():
        domain[key] = colors[:]
    solve(domain, adj)
    print(domain)
```

Output:



```
In [1]: runfile('C:/Users/Srinidhi/Desktop/graph_coloring.py', wdir='C:/Users/Srinidhi/
Desktop')
{'sa': ['g'], 'q': ['r'], 'nsw': ['b'], 'nt': ['b'], 'wa': ['r'], 'v': ['r'], 't': ['r']}
System Time :
11:57:27
```



Result:

Thus, the graph colouring problem is implemented by satisfying the given constraint.